

API CA Prep Day 4: Production-like Config & Swagger Docs

Table of Contents

1. Core Requirements
2. `app.js` Cleanup & Route Refactor
3. Database Sync & Seeding
4. Swagger Setup & Requirements
5. Response Schemas & Auth
6. Final Checks & Reflection


1. Core Requirements

Today you're finishing the last major step before the assessment:

- Replace the mock app with your real `app.js`
- Use versioned API routes (`/api/v1/...`)
- Switch from test DB to real `.env` config
- Document the entire API using **Swagger Autogen**
- Seed your real DB with EventTypes (once)
- Reflect on your readiness

2. `app.js` Cleanup & Route Refactor

Time to clean up the Express Generator leftovers and wire up the real app.

 Remove the following:

```
// We dont use a view engine
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

// We dont use form data, cookies, or expose static content
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// You will change these to your own routes (/api/v1/auth and /api/v1/events)
app.use('/', indexRouter);
app.use('/users', usersRouter);

// We have our own error handling middleware
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};
```

```
// render the error page
res.status(err.status || 500);
res.render('error');
});
```

NOTE: Remember to also remove unused dependencies (`require(...)`)

✓ Configure your real app (app.js):

- Use `/api/v1` as the prefix for all routes
- Register routes, middleware, and Swagger UI
- Ensure middleware order:
 1. `express.json()`
 2. Route handlers
 3. Centralized error handler
- Mount Swagger at `/docs`

3. Database Sync & Seeding

Update your Sequelize sync to use:

```
sequelize.sync({ alter: true })
```

This will apply model changes **without wiping your data** between restarts.

✓ Seed Event Types (but only once)

You **should not** reinsert data every time the app runs.

Use a simple check to see if any EventTypes exist before seeding (example below).

```
const count = await db.EventType.count()
if (count === 0) {
  await db.EventType.bulkCreate([
    { id: 1, name: 'Conference' },
    { id: 2, name: 'Meetup' },
    { id: 3, name: 'Workshop' },
    { id: 4, name: 'Seminar' }
  ])
}
```

4. Swagger Setup & Requirements

Use **Swagger Autogen** (as done in previous lessons) to document your full API.

 Link to [repo](#) from class.

Each route should include:

- Summary & description
- Request body schema (if applicable)
- **Success** and **fail** response examples (JSend format)
- Token auth config (where applicable)

Document:

Endpoint	Auth	Description
/api/v1/auth/signup	✗	Register new user
/api/v1/auth/login	✗	Log in and get JWT
/api/v1/events (GET)	✗	Get all events
/api/v1/events/mine	✓	Get events for current user
/api/v1/events (POST)	✓	Create event
/api/v1/events/:id (PUT)	✓	Update event (if user owns it)

5. Response Schemas & Auth

✓ JSend: Success Example

```
{
  "status": "success",
  "data": {
    "id": 3,
    "title": "Test Event 3",
    "date": "2025-05-01",
    "location": "Oslo",
    "eventId": 1,
    "userId": 1
  }
}
```

✗ JSend: Fail Example

```
{
  "status": "fail",
  "data": "Date must be in the future"
}
```

Auth in Swagger

Protected endpoints must use **Bearer tokens**.

Follow the [Swagger Autogen token example](#) to enable this.

6. Final Checks & Reflection

✓ Final Checklist

- ☐ Run your real app using `npm start`
- ☐ Routes are mounted under `/api/v1`
- ☐ Swagger UI works at `/docs`
- ☐ All routes are documented clearly
- ☐ JWT can be used via the "Authorize" button
- ☐ JSend format is consistent
- ☐ DB uses `.env` config and seeds correctly

To live test the application you can do the following:

1. Signup
2. Login (copy token)
3. Use token to create an event
4. Use token to see your events (`events/mine`)
5. See all events (without token)
6. Update your event
7. If you want, you can signup as a new user and test the flow again - this way you can see the different responses.

🗨 Reflection: Are You Ready for the CA? (and beyond)

Take a moment to reflect on where you're strong - and where you still feel uncertain.

NOTE: Some of these reflections are not relevant for what's coming in this CA (e.g., testing services, creating models) but reflect on them to gauge how much you have learnt in this project.

Here are some questions to guide your thinking:

- **Models & Sequelize**
 - Can I define models with validation and associations from scratch?
 - Do I understand how syncing and seeding works?
- **Service Layer**
 - Can I write service methods that use Sequelize properly?
 - Am I confident testing these methods?
- **JWT Auth**
 - Do I understand how JWTs are created and verified?
 - Could I explain how password hashing and token verification work?
- **Routes & Middleware**
 - Can I protect routes using middleware?
 - Do I know when to return different error types (400 vs 401 vs 403)?
 - Am I comfortable with centralized error handling?

- **Swagger & Docs**

- Do I know how to document APIs clearly?
- Can I write good examples that help others use my API?

- **Debugging & Testing**

- Am I confident using Supertest to verify routes?
- Do I know how to troubleshoot when something breaks?